A LAGRANGIAN RELAXATION ALGORITHM FOR THE MULTI - RESOURCE GENERALIZED ASSIGNMENT PROBLEM

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

N. V. RAO

87008

to the

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

MAY, 1985

87588

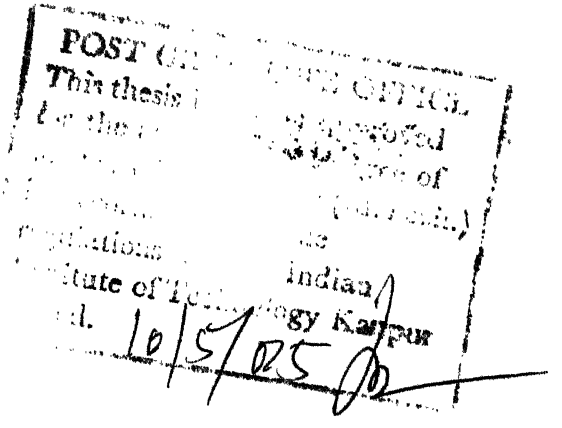
IMEP-1985-M-RAO-LAG

CERTIFICATE

This is to certify that the work entitled, A LAGRANGIAN RELAXATION ALGORITHM FOR THE MULTI-RESOURCE GENERALIZED ASSIGNMENT PROBLEM, by Shri N.V. Rao, has been carried out under my supervision and has not been submitted elsewhere for a degree.

RKAhuja

(R.K. Ahuja)
Lecturer
Industrial and Management Engg.
Indian Institute of Technology
Kanpur 208 016



ACKNOWLEDGEMENTS

I wish to express my gratitude and sincere thanks to my thesis supervisor Dr. R.K. Ahuja for his responsible and useful guidance throughout the course of this work.

I am thankful to my friend Mr. VVSN Murthy, for assisting me in many ways throughout my stay at IIT Kanpur.

I express my deep sense of affection for my friends Mr. Y.V. Ramana and Mr. P. Sridhar Gopal for their constant inspiration and help throughout my studies.

I thank Swami Anand Chaitanya for his excellent typing and Mr. Buddhi Ram Kandiyal for his neat cyclostyling work.

N. V. Rao

CONTENTS

Chapter		Page
I.	INTRODUCTION	1
	1.1 Introduction	1
	1.2 Statement of the Problem	3
	1.3 Applications	5
	1.4 Outline of the Thesis	6
II.	LITERATURE REVIEW	9
	2.1 Introduction	9
	2.2 Literature Review	9
	2.3 Lagrangian Relaxation Concepts	15
III.	THE BRANCH AND BOUND ALGORITHM	18
	3.1 Introduction	18
	3.2 The Branch and Bound Algorithm	18
	3.3 Numerical Example	34
	3.4 Computational Results	37
	3.5 Concluding Remarks	40
	REFERENCES	43

ABSTRACT

The Multi Resource Generalized Assignment Problem (MRGAP) is to determine an optimal allocation of tasks to agents such that the requirements of tasks allocated to agents do not exceed the available resources with agents. The MRGAP is a generalization of the well-known generalized assignment problem in the sense that each agent possesses multiple resources and allocation of tasks must satisfy each resource limitation. We describe several applications of MRGAP and develop a branch and bound algorithm based on lagrangian relaxation concepts. A novel feature of the algorithm is that the lagrangian multipliers are computed only once and these multipliers are used to compute the lower bound for each subproblem. The proposed algorithm was coded and tested extensively on randomly generated problems of different parameters. Results of these computations are quite encouraging. The algorithm can solve problems with 1000 0-1 variables and 6 resources optimally in less than 60 seconds and problems with 8000 variables and 6 resources within 1 percent of optimality in less than 3 minutes.

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION:

The formulation of models with different mathematical structures and the development of algorithms to exploit these structures continues to be a productive direction for research in mathematical programming. During the last few years, several similarly structured integer programming models, namely, weighted assignment models, have been developed for a number of diverse applications. In this thesis, we deal with one such mathematical model, which comes into the category of weighted assignment models, Multiple Resource Generalized Assignment Model, which will be referred to as MRGAP, hereafter. This problem is a known NP-complete problem and we develop a branch and bound algorithm for this problem based on lagrangian relaxation concepts.

In the classical assignment model [10], we have agents and tasks are done by these agents. The aim is to find out the optimal pairings of agents and tasks. The optimality is determined by a criterion function like cost etc. Each task is to be assigned to a single agent and each agent should be

given only one task. These constraints restrict the model to a limited domain of applications like assigning air crew to the flights, assigning jobs to machines or men and space to departments etc. However, in many practical situations the number of tasks to be done are more than the number of agents and accordingly more than one task should be given to each agent. This leads to a more useful model which allows for more than one task to be assigned to each agent, provided these tasks would not require more of some resource than is available with the agent. This type of model, which is given the name Generalized Assignment Problem (GAP) by Ross and Soland [11], has many applications like assigning jobs to computers in a computer network, assigning software development tasks to programmers and scheduling the variable length television commercials into time slots etc.

From the applications stand point, a more useful model would be the one which allows for more than one resource with each agent and which allows the tasks to be completed in two or more stages, that is, the tasks will be completed by the agents at different levels. These situations generally arise in sophisticated systems where everything (for example, manpower, equipment, capacity, etc.) is considered to be very critical and has to be taken care of while finding the best assignments. These models are termed as Weighted Assignment Models (WAM) in literature.

In this thesis, we consider a special case of the WAM in which tasks are to be finished at a single level, that is, if a task is assigned to an agent, it will be finished completely by that agent at a time. Each agent will have more than one resource which will be consumed when a task is assigned to it. The novel features of our algorithm are that lagrangian relaxation concepts are used to obtain tight lower bounds and the lagrange multipliers are calculated only at the root node and are used throughout the branch and bound tree. The LIFO treatment of candidate problems had been employed in the branch and bound algorithm. Numerical investigations with the algorithm are presented in detail.

1.2 STATEMENT OF THE PROBLEM:

The notations that are used throughout this thesis have been given below for the sake of completeness.

I : Set of agents,

J : Set of tasks

m : $|I|$,

n : $|J|$,

p : Number of resources associated with each agent,

$$x_{ij} = \begin{cases} 1, & \text{if task } j \text{ is assigned to agent } i \\ 0, & \text{otherwise} \end{cases}$$

b_i^k : The amount of resource k available with the agent i,

c_{ij} : Cost of assigning task j to agent i,

a_{ij}^{k} : Amount that will be consumed of resource k when task j is assigned to agent i.

In this thesis, we have considered a model where a set of tasks must be divided among a set of agents and each task must be completed by only one agent. Once a task is started by an agent, it will be finished completely by that agent at one time. When an agent completes a task, the resources possessed by the agent are consumed and a certain amount of cost is incurred. The model is used to determine the best assignment of tasks to agents so as to minimise total system cost while satisfying the agent resource constraints.

Mathematically, the model can be represented as follows:

$$\text{Min} \quad \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$$

s.t.

$$\sum_{i \in I} x_{ij} = 1, \quad \forall\ j \in J, \tag{1.1}$$

$$\sum_{j \in J} a_{ij}^{k} x_{ij} \leq b_{i}^{k}, \quad \forall\ i \in I, \forall\ k = 1,\ldots,p, \tag{1.2}$$

(P)

$$x_{ij} = (0, 1), \quad \forall\ i \in I, \quad \forall\ j \in J. \tag{1.3}$$

This problem becomes similar to the Generalized Assignment Problem when the number of resources possessed by each agent is equal to one.

1.3 APPLICATIONS:

The MRGAP has applications in determining optimal assignments of tasks to agents when the number of tasks is larger than the number of agents and the number of resources possessed by each agent is more than one. For example, the agents may be the dock yards and the tasks to be done could be the damaged ships. The resource constraints could be the number of berths available, number of equipment hours available and the number of man hours available etc. The objective function could be the minimisation of the cost of transportation of damaged ships to the yards i.e. the ships have to be transported to the yards and the cost of transportation vary with the distance and size of the ship. A similar application could be in the case of railway workshops and wagons.

Alternatively, the agents could be the district headquarters and the tasks are the schools in that area and our aim is to assign the schools to these headquarters, ignoring the geographic boundaries of the districts, such that the sum of weighted distances of these schools from their respective headquarters will be minimum. The resource constraints for each headquarter could be the numbers of children it can control, number of inspector days available with it and the amount of funds etc. Similarly, police districting can also be formulated as MRGAP. Here the agents are the police stations and the tasks are the localities and the aim is to assign the

localities to the police stations so as to minimize the sum of distances of the localities from their respective police stations, while taking care of resource constraints like the number of police, number of crimes that can be controlled and the amount of funds available for each station.

Another important application of MRGAP is the optimal assignment of jobs to computers in a time sharing computer network environment. The agents are the computers and the tasks are the various programs that are to be executed on these computers. Each job may take different processing times on different computers and it may cost differently on different computers. Also, the various types of memory requirements may change from computer to computer, particularly when the computers are of different categories like super, main frame or mini computers etc. So the resource constraints will be the amount of memory available of each type and input, output capacities etc. and the amount of processing time available to each computer. The objective function could be the minimization of the cost of processing a given set of jobs.

1.4 OUTLINE OF THE THESIS:

A brief outline of the thesis is given in this section.

In Chapter II, we survey the existing literature in the field of Generalized Assignment Models. We present the salient features of the Ross and Soland's algorithm [11] for

the generalized assignment problem which is a special case of our model and discuss the advantages and disadvantages of his approach to our model. We also give the applications of lagrangian relaxation from the literature. Also, a brief account of the concepts of lagrangian relaxation and subgradient optimisation is presented.

In Chapter III, we develop the algorithm for MRGAP. The algorithm is developed on the basis of branch and bound approach. In this algorithm, we use the lagrangian relaxation concepts to a limited extent which improves the performance of the algorithm substantially. Another feature of the algorithm is that the lagrangian multipliers are not updated in the latter part of the branch and bound tree. The search method adopted in the algorithm is depth first search. It has been found empirically that depth first search will yield a good feasible solution early in the branching process.

To test the computational performance of the algorithm, a FORTRAN program was written and tested on a number of randomly generated problems. Computational results showed that the algorithm can solve reasonably large sized problems. It solves a 1000 0-1 variables and 6 resources problem in about 60 seconds. These computational results are presented in detail. A numerical example was also given in this chapter.

Computational results have also been presented for a ε - optimal algorithm which is useful for solving very large

sized problems. It was found that with a value of 0.01 for ε, the algorithm has been able to solve a 8000 0-1 variables and 6 resources problem in about 3 minutes.

CHAPTER II

LITERATURE REVIEW

2.1 INTRODUCTION:

In this chapter, we present the previous work done in the area of the assignment models. We also present, in detail, the Ross and Soland's algorithm for GAP and discuss its merits and demerits to apply their approach to our model. We then give a brief review of literature on the lagrangian relaxation and subgradient optimisation. We also briefly explain the concepts of lagrangian relaxation and subgradient optimisation at the end.

2.2 LITERATURE REVIEW:

The development of algorithms for various assignment models has been a potential area for research for a long time and continues to be one. In 1956, KUHN [10] developed an algorithm for the classical assignment model which is known as the Hungarian method. Subsequently, several alternate algorithms have been suggested for this problem. Barr, Glover and Klingman [2] have designed an alternating path basis algorithm to this model which does not consider all feasible bases for progressing to an optimal basis.

Hung and Rom [8] suggested an algorithm based on a scheme of relaxing the given problem into a series of simple network flow problems for each of which an optimal solution is easily obtained.

Another model, namely Generalized Assignment **Problem (GAP)**, which is a generalization of the classical assignment problem, has been formulated by Ross and Soland [11] in 1975. The mathematical formulation of this model is as follows:

$$\text{Min} \quad \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}$$

s.t.

$$\sum_{i \in I} x_{ij} = 1, \quad \forall\, j \in J \tag{2.1}$$

(GAP)

$$\sum_{j \in J} a_{ij} x_{ij} \leq b_i, \quad \forall\, i \in I \tag{2.2}$$

$$x_{ij} = (0,1), \quad \forall\, i \in I, \quad \forall\, j \in J. \tag{2.3}$$

This problem takes on more of the appearance of the classical assignment problem when each constraint (2.2) is scaled by dividing both sides of the inequality by the right-hand side value to obtain,

$$\sum_{j \in J} K_{ij} x_{ij} \leq 1, \quad \forall\, i \in I \tag{2.4}$$

where, $K_{ij} = a_{ij}/b_i$

It is apparent that the classical assignment problem is a special case of GAP in which $K_{ij} = 1$ for all $i \in I$ and $j \in J$ and $m = n$.

Ross and Soland have suggested a branch and bound based exact algorithm for GAP. The main features of their algorithm are that the algorithm solves a series of continuous value knapsack problems in order to obtain tight lower bounds. The validity of this lower bound has been proved by viewing it as a specific application of lagrangian relaxation where the constraints (2.1) in GAP are relaxed. The objective function value of this relaxed version will provide a lower bound for the optimal solution of the original problem GAP. This approach of lagrangian relaxation is commonly used for constructing tight lower bounds in integer programming problems.

Another characteristic feature which has been exploited in their algorithm is the LIFO treatment of candidate problems in the branch and bound tree. This is beneficial both in terms of core storage and solution time requirements. With this arrangement only those restrictions associated with the current relaxation need to be stored. It has been observed by them empirically that LIFO approach yields a good feasible solution early in the branching process.

Later, other algorithms have been suggested for GAP. Fisher, Jai Kumar and Van Vassenhove [4] have developed a multiplier adjustment method and Klastorin [9] developed a heuristic method for the same problem to solve large sized problems.

A general formulation of these models have been given by Ross and Zoltners [12] who gave a general name weighted Assignment Models to this type of problems. The mathematical formulation of this general model is as follows:

$$\text{Min} \sum_{i\varepsilon I} \sum_{j\varepsilon J} \sum_{h\varepsilon H_{ij}} c_{ijh}\, x_{ijh}$$

s.t.

$$\text{(WAP)} \qquad \sum_{i\varepsilon I} \sum_{h\varepsilon H_{ij}} x_{ijh} = 1, \quad \forall\, j \in J, \tag{2.4}$$

$$r_i^k \leq \sum_{j\varepsilon J} \sum_{h\varepsilon H_{ij}} a_{ijh}^k\, x_{ijh} \leq b_i^k, \quad \forall\, i \in I, \forall\, k \in K, \tag{2.5}$$

$$x_{ijh} = (0,1), \quad \forall\ i \in I, \forall\, j \in J, \forall\, h \in H_{ij}. \tag{2.6}$$

This model represents problems with the following characteristics: A set of tasks must be divided among a set of agents, and each task must be completed by only one agent at one of the several predetermined levels. When an agent completes a task at some level, the resources possessed by the agent are consumed and a certain amount of cost is incurred. The model is used to determine the best assignment of tasks to agents and the task completion levels.

It is clear that the classical assignment problem, GAP and MRGAP are the special cases of WAP. When the tasks are completed at only one level, that is, if a task is assigned to some agent, it will be finished completely by that agent at a time and when the number of resources associated with each agent is equal to one, the resulting problem will be GAP.

Ross and Zoltners also discussed the relation of this model to the transportation models, knapsack models, facility location models, transignment models etc. They have provided applications for various models described by them and the information about their algorithms.

The model considered in this thesis (MRGAP) is also a special case of WAP where the tasks will be completed at only one level. The number of resources associated with each agent can be more than one.

The difficulty in applying the approach of Ross and Soland to our model is that the knapsack problems will become multi-dimensional in our problem. Each knapsack problem will have p constraints where p is the number of resources associated with each agent. We are not aware of any algorithm for the multi-dimensional knapsack problem.

Because of this difficulty, a different approach of the lagrangian relaxation has been adopted in this thesis where instead of constraints (1.1), as has been done by Ross and Soland, we relax constraints (1.2) in (P). The lagrangian relaxation concepts and their applications in literature have been excellently reviewed by Fisher [3] in 1981. He gave details of the basic constructions of lagrangian relaxation and its applications. He also discussed the criteria to decide between the competing relaxations as there are more than one way of

applying lagrangian relaxation to a model. He also reviewed the merits and demerits of various methods of updating the lagrangian multipliers and finally explained how these concepts can be used in a branch and bound algorithm.

The lagrangian relaxation has been used successfully to various models in the literature. Held and Karp [6] have first applied this principle in their highly successful branch and bound algorithm for travelling salesman problem. Subsequent to the successful work of Held and Karp, many applications of lagrangian relaxation have appeared in the literature. Shogan [13] applied these concepts to the resource constrained, capacitated minimum cost spanning tree problem. Gavish [5] applied lagrangian relaxation to a computer network design problem.

The updating of lagrangian multipliers is the critical part of the lagrangian relaxation procedure. Obtaining the optimal multipliers, that is, the one which gives the highest lower bound, is highly time consuming owing to large number of linear inequalities involved. So a new method, namely subgradient optimisation is being used widely in the literature with tremendous success. In this method, the aim is to calculate a suboptimal set of multipliers, by approximating the subgradient as the steepest ascent gradient to the optimal solution. A detailed discussion of subgradient optimisation is given by Held, Wolfe, Crowder [7]. The proof of the validity

of the subgradient optimisation procedure is also given in this paper.

2.3 LAGRANGIAN RELAXATION CONCEPTS:

In this section, we will briefly explain the concepts of lagrangian relaxation and its validity.

We begin with a combinatorial problem formulated as the integer program:

$$
\begin{aligned}
& Z = \min c\,x \\
(PL) \qquad & Ax \leq b \\
& x \geq 0 \text{ and integral}
\end{aligned}
$$

We assume that the constraints of (PL) have been relaxed to make it easy to solve the lagrangian problem

$$
\begin{aligned}
& Z_D(v) = \min (cx + v(Ax - b)) \\
(LR_v) \qquad & x \geq 0 \text{ and integral}
\end{aligned}
$$

where, v is the set of lagrangian multipliers.

Now, we show that the objective function value of (LR_v) will be a lower bound on the optimal value of Z for any particular set of lagrangian multipliers, i.e. we want to show that $Z_D(v) \leq Z$ for any v.

When the constraints of the type $Ax \leq b$ are relaxed, we require that $v \geq 0$.

Now, let x^* be the optimal solution to (PL), then

$$Z_D(v) \leq (c\,x^* + v(Ax^* - b)) \leq Z$$

where the first inequality follows from the definition of $Z_D(v)$ and the second inequality from $Z = c\,x^*$, $v \geq 0$ and $Ax^* - b \leq 0$. Similarly, for $Ax \geq b$, we require $v \leq 0$ for $Z_D(v) \leq Z$ to hold.

The fact that $Z_D(v) \leq Z$ allows (LR_v) to be used in place of (LP) to provide lower bounds in a branch and bound algorithm for (PL).

Determining v:

As a tighter lower bound will help the branch and bound algorithm in pruning the nodes effectively, our aim is to determine v such that $Z_D(v)$ will be maximum. In other words, our aim is to solve the problem,

$$(D) \qquad Z_D = \max_{v} \; Z_D(v) \qquad v \geq 0$$

There are several methods to determine the optimal v, but according to Fisher [3], the subgradient method is the simple to program and also requires minimum computations and gives very good values of v. We briefly explain the subgradient method here.

In the subgradient method, given an initial set v^o, a sequence $\{v^r\}$ is generated by the rule

$$v^{r+1} = v^r + t_r\,(Ax^r - b)$$

where x^r is an optimal solution to (LR_v) and t_r is a positive scalar step size.

The step size used most commonly in practice is

$$t_r = \frac{\lambda_r (Z^* - Z_D(v^r))}{|| Ax^r - b ||^2}$$

where λ_r is a scalar satisfying $0 < \lambda_r \leq 2$ and Z^* is an upper bound on Z_D.

Unless we obtain a v^r for which $Z_D(v^r)$ equals the cost of a known feasible solution, there is no way of proving optimality in the subgradient method. To resolve this difficulty, the method is usually terminated upon reaching an arbitrary iteration limit.

CHAPTER III

THE BRANCH AND BOUND ALGORITHM

3.1 INTRODUCTION:

In this chapter, we develop the branch and bound algorithm for MRGAP based on the lagrangian relaxation concepts. Some of the important features of the algorithm are that (1) the lagrangian multipliers are calculated only at the root node, (2) the lower bounds are not calculated at all the nodes of the branch and bound tree.

In Section (3.2) we present the flow chart of the proposed algorithm and a step-by-step explanation of the flow chart. In Section (3.3), a numerical example is given. In Section (3.4) computational performance of the algorithm is presented in detail. The results of ε-optimal procedure are also presented. Finally, concluding remarks are given in Section (3.5).

3.2 THE BRANCH AND BOUND ALGORITHM:

The branch and bound algorithm developed in this thesis has a special feature that the bounds are calculated using lagrangian relaxation concepts and some additional characteristics have been exploited to reduce the computations at each node.

The basic idea behind any branch and bound algorithm is to investigate all the feasible solutions by partitioning the solution set into subsets, by a process known as branching. A lower bound is calculated for the objective function value of the solutions in each subset. If the lower bound is greater than or equal to the objective function value of the incumbent solution, that is, all the solutions in the subset are worse than the incumbent solution then further branching from this subset is stopped. This is called pruning. The partitioning continues until the objective function value of the incumbent solution is lower than any bound for any subset. The algorithm terminates when all the subsets are pruned.

In the discussion to follow about the branch and bound algorithm, Z denotes the objective function value of the incumbent, the best known feasible solution of (P). Z is initially set to the objective function value of the solution obtained by assigning each task to the agent with maximum cost ignoring the resource constraints.

In the branch and bound tree, the root node represents the problem (P). We will partition the solution set of this problem over a separation variable, let us say $x_{\alpha\beta}$ into two subsets, one subset containing all the solutions with $x_{\alpha\beta}$ equal to one and the other containing solutions with $x_{\alpha\beta}$ equal to zero. These resulting problems of partition are similar to (P) but additional restrictions are placed on the solution

matrix x. These problems are of reduced size and are called candidate problems. Now, our aim is to calculate the lower bounds for each of these two candidate problems. Let (P^h) represents a candidate problem.

The basic concept in the bounding procedure of the algorithm is best illustrated with reference to the initial relaxation of problem (P). There are two natural relaxations of (P), one is by relaxing the constraints (1.1) and the other by relaxing constraints (1.2) in (P). The mathematical formulations of these relaxations are given below:

When constraints (1.1) are relaxed, the model becomes,

$$Z_1(u) = \text{Min} \sum_{i\varepsilon I} \sum_{j\varepsilon J} c_{ij} x_{ij} + \sum_{j\varepsilon J} u_j \left(1 - \sum_{i\varepsilon I} x_{ij}\right)$$

or, equivalently,

$$Z_1(u) = \text{Min} \sum_{i\varepsilon I} \sum_{j\varepsilon J} (c_{ij} - u_j) x_{ij} + \sum_{j\varepsilon J} u_j$$

s.t.

$$(LR_1) \qquad \sum_{j\varepsilon J} a_{ij}^k x_{ij} \leq b_i^k, \ \forall\, i \in I, \ \forall\, k = 1,\ldots p$$

$$x_{ij} = 0 \text{ or } 1, \ \forall\, j \in J, \ \forall\, i \in I.$$

Thus (LR_1) separates into a series of multi-dimensional knapsack problems, one for each $i \in I$. This is a hard problem if an exact solution is desired. However, a lower bound can be obtained by solving m linear programming problems of size p x n.

The second relaxation (LR_2) is obtained by relaxing constraints (1.2) with $v \geq 0$

$$Z_2(v) = \text{Min} \sum_{i\varepsilon I} \sum_{j\varepsilon J} c_{ij} x_{ij} + \sum_{i\varepsilon I} \sum_{k\varepsilon K} v_i^k \left(\sum_{j\varepsilon J} a_{ij}^k x_{ij} - b_i^k \right)$$

or, equivalently,

$$Z_2(v) = \text{Min} \sum_{j\varepsilon J} \sum_{i\varepsilon I} \left(c_{ij} + \sum_{k\varepsilon K} v_i^k a_{ij}^k\right) x_{ij} - \sum_{i\varepsilon I} \sum_{k\varepsilon K} v_i^k b_i^k$$

s.t.

(LR_2)

$$\sum_{i\varepsilon I} x_{ij} = 1, \quad \forall\, j \varepsilon J$$

$$x_{ij} = 0 \text{ or } 1, \quad \forall\, j \varepsilon J, \forall\, i \varepsilon I$$

Now, one may discuss the merits and demerits of these relaxations. To evaluate a relaxation, we have to consider two properties: The sharpness of the bounds and the amount of computations required to obtain these bounds. Usually, selecting a relaxation involves a tradeoff between these two properties, It is generally difficult to determine whether a relaxation with sharper bounds but greater computational time requirements will result in a branch and bound algorithm with better overall performance. In our problem, the relaxation (LR_1) needs solving m multidimensional knapsack problems at each iteration or alternatively, m linear programming problems. A multidimensional knapsack problem [14] is one with the same basic construction as the well known knapsack model but have more than one constraint. This problem is a well known NP-complete problem and we are not aware of any algorithm for solving it. Also, it may

not be computationally feasible to solve m linear programming problems at each iteration.

In contrast, solving (LR_2) requires solving problem of the type

$$\text{Min } (c_{ij} + \sum_{k=1}^{p} v_i^k a_{ij}^k) x_{ij}$$

$$\sum_{i \varepsilon I} x_{ij} = 1$$

$$x_{ij} = 0 \text{ or } 1, \forall\ i \varepsilon I$$

for each $j \varepsilon J$.

To solve this, we have to find an agent among I, which is having minimum $(c_{ij} + \sum_{k=1}^{p} v_i^k a_{ij}^k)$ value for each $j \varepsilon J$ and the corresponding decision variable is made equal to one. This requires less computations, of the order of mnp, when compared to (LR_1). Because of this great difference in computational time requirements we adopt relaxation (LR_2) in our branch and bound algorithm, though it produces feeble lower bounds relative to (LR_1).

Now, we will see how the lagrangian relaxation principles increase the efficiency of the branch and bound algorithm.

Let Z_{opt} and $Z_2(v)$ denote the optimal objective values for problems (P) and (LR_2), respectively. Clearly, $Z_2(v) \leq Z_{opt}$, that is, $Z_2(v)$ is a lower bound on Z_{opt} for any non-negative v, as shown in Chapter II. A traditional branch and bound approach

might use Z_0 as a lower bound for Z_{opt}, that is, compute a lower bound on Z_{opt}, that is, compute a lower bound on Z_{opt}, by simply ignoring the resource constraints and solving the resulting problem. However, a branch and bound algorithm based on L-relaxation seeks to increase the likelihood of fathoming a node by searching for the greatest possible lower bound, that is, the objective will be to solve

$$Z_L = \underset{v}{\text{maximise}} \; Z_2(v)$$

subject to

(PM)

$$v_i^k \geq 0, \quad \forall\, i \in I, \forall\, k = 1, \ldots, p.$$

There are other benefits of this approach in addition to the fathoming of nodes on the basis of lower bounds. In particular, instead of simply solving (P_0), the attempt to solve (PM) involves the solution of a series of problems of the form (LR_2) with a different value of v each time. This increases the likelihood of obtaining a feasible solution which may be better than the incumbent value, thus increasing the likelihood of fathoming the nodes in the subsequent part of the branch and bound tree.

Fig. 1 contains the flow chart of the branch and bound algorithm. A more detailed discussion of each of the steps follows: Let S^v denote an m x p matrix whose S_{ik} is the value $\left(\sum_{j \in J} a_{ij}^k x_{ij} - b_i^k \right)$ evaluated at v. A sub-problem in the branch and bound algorithm consists of three sets of variables:

variables whose values are fixed to zero, variables whose values are fixed to one and the variables which are unassigned. For simplicity of presentation, let us assume that two matrices y^o and y^1 are associated with each sub-problem defined as below:

$$y^o_{ij} = \begin{cases} 1, & \text{if } x_{ij} \text{ is fixed to zero} \\ 0, & \text{otherwise, } \forall\, i \in I, \forall\, j \in J \end{cases}$$

$$y^1_{ij} = \begin{cases} 1, & \text{if } x_{ij} \text{ is fixed to one} \\ 0, & \text{otherwise, } \forall\, i \in I, \forall\, j \in J \end{cases}$$

These are used to represent the additional restrictions on the candidate problem at any instant of the branch and bound tree.

Step 1: In the branch and bound tree the first node, i.e. the root node, represents the initial relaxation of the problem (P). The lagrangian multiplier set is calculated at this node using subgradient optimisation and will be used throughout the tree.

To start with, all the lagrangian multipliers are made equal to zero. In lagrangian relaxation, we try to increase the cost of allocating the task to those agents which are overloaded, they making it possible to transfer some of the tasks to other agents which are under utilised. We do this by adding a factor to the actual cost of allocating a task. This factor is $\sum_{k=1}^{p} v_i^k a_{ij}^k$ as seen in the formulation (LR_2), where v_i^k is the lagrangian multiplier corresponding to the constraint on k-th

resource of i-th agent. It is natural to start with a value of zero to all v_i^k's and then increasing v's of those agents which are overloaded.

Now, our aim is to obtain a good suboptimal, if not optimal, values for the lagrangian multipliers using subgradient optimisation. For this, we make an attempt to solve (PM). But solving (PM) optimally is impractical owing to the large number of linear inequalities represented in the problem (LR_2), hence an attempt to solve (PM) will be made through an iterative procedure known as subgradient optimization [7], subgradient optimisation is so named because S^v is a subgradient of the objective function for (PM) at the point v, and the procedure optimally uses this subgradient as if it were a gradient pointing in the direction of steepest ascent. Iteration r of the subgradient optimisation consists of the following steps (1) movement to the new v^r obtained recursively from v^{r-1} (2) solution of the L-relaxation using new v (3) deciding whether to perform another iteration or not. The hope is that the sequence (v^r) will yield a good suboptimal solution to (PM). Each step of the subgradient procedure is explained below in detail.

Given the lagrangian multiplier set, the relaxed problem (LR_2) is solved by assigning each task to that agent with minimum $(c_{ij} + \sum_{k=1}^{p} a_{ij}^k v_i^k)$ value and whose capacities are not less than the consumption of the resources by this task.

After (LR_2) is solved with a particular set of v, we try to revise the values of lagrangian multipliers to obtain a bigger objective function value for (LR_2) which is a lower bound on the objective function value of (P). Updating lagrangian multipliers is the most complicated procedure in the branch and bound algorithm. The procedure is based on a proof [7] that if the non-negative scalar t^r is sufficiently small, then

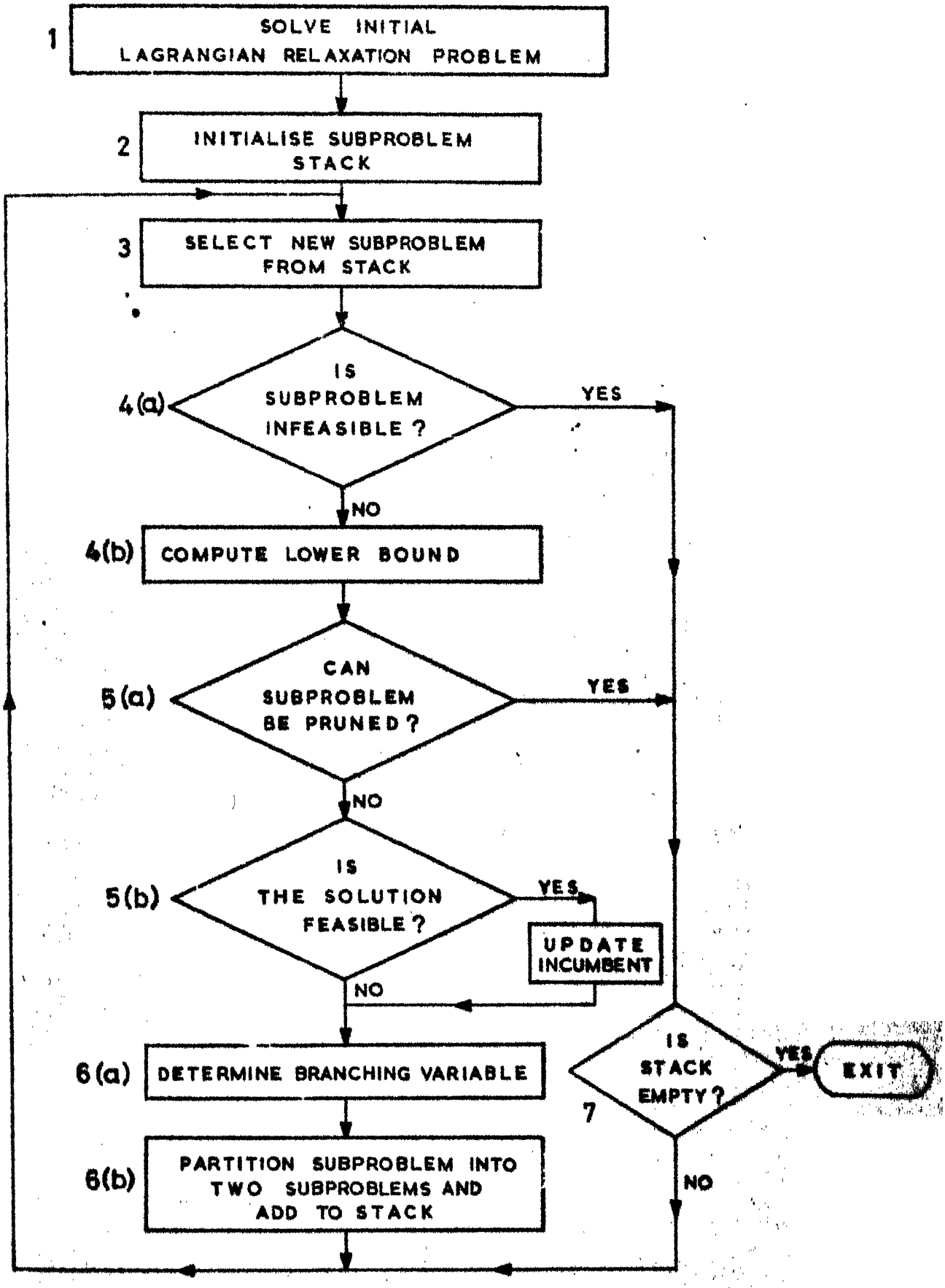
$$v^{r+1} = \max [0, (v^r + t^r S^v)]$$

is closer than v^r to the optimal solution of (PM), although it does not guarantee an improvement in the lower bound. The step size t^r most commonly used in practice is,

$$t^r = \frac{\lambda_r (Z^* - Z_2(v^r))}{||Ax^r - b||^2}$$

where λ_r is a scalar satisfying $0 < \lambda_r < 2$ and Z^* is an upper bound on the optimal solution value of the problem (P). The value of λ_o can be obtained empirically with some experimentation and it is halved whenever Z_2 has failed to improve in some fixed number of iterations.

Now, a decision should be taken as to when the subgradient optimisation procedure is stopped. There is no way of proving the optimality of v's in the subgradient method unless we obtain a v such that $Z_2(v)$ is equal to a known feasible solution value. To resolve this difficulty the method is usually terminated upon satisfying one of the following conditions.


1
SOLVE INITIAL LAGRANGIAN RELAXATION PROBLEM
2
INITIALISE SUBPROBLEM STACK
3
SELECT NEW SUBPROBLEM FROM STACK
4(a)
IS SUBPROBLEM INFEASIBLE ?
YES
NO
4(b)
COMPUTE LOWER BOUND
5(a)
CAN SUBPROBLEM BE PRUNED ?
YES
NO
5(b)
IS THE SOLUTION FEASIBLE ?
YES
UPDATE INCUMBENT
NO
6(a)
DETERMINE BRANCHING VARIABLE
6(b)
PARTITION SUBPROBLEM INTO TWO SUBPROBLEMS AND ADD TO STACK
7
IS STACK EMPTY ?
YES
EXIT
NO

(a) The scalar step size $t^r < \varepsilon_1$

(b) The scalar $\lambda_r < \varepsilon_2$

where ε_1 and ε_2 are very small quantities or,

(c) the number of iterations $r > D$, where D is an arbitrary iteration limit.

Step 2.3.7: The stack contains the list of unfathomed problems. A subproblem in the stack is represented by a vector of information (y^l, y^o, v, Z^h, S^v), where v is not necessarily the optimal solution to problem (PM) but is such that Z^h is the greatest known lower bound on the optimal solution value of the subproblem (P^h).

Initially, in Step (1), we solve the problem (LR_2), at the root node of the branch and bound tree, which results from ignoring the resource constraints in problem (P). Then the partitioned subproblems from this root node are stored in the stack. The storing of subproblems is done in such a way that it needs minimum effort when a variable is selected to branch out from a particular node.

In selecting a new subproblem from the stack, the subproblem is chosen according to the depth first rule, that is, the chosen subproblem is the one which is the latest addition to the stack, or equivalently, the one on the top of the stack. This approach results in a significant reduction in the computer storage, since only the details of the latest

subproblem have to be stored at any instant of the algorithm. However, if the stack is empty (Step 7), that means all subproblems have been pruned and the incumbent solution matrix is the optimal solution of (P).

There are other strategies existing for selecting a subproblem from the stack. The most appealing alternative is the best-bound rule, which selects the new subproblem as the one with the lowest lower bound among the unfathomed subproblems. Such an approach would result in lower percentage errors, when the algorithm is forced to terminate prematurely due to excessive execution times, because we are considering the most promising subproblems first. Inspite of this, the depth first rule was chosen because the best-bound rule requires lot of computations to update the list of subproblems and also because it is observed empirically that the depth first rule will yield a good feasible solution early in the branching process. Of course, there is no guarantee that the depth first rule is the best choice.

Step 4: (Computing the lower bound) An important feature of our branch and bound algorithm is that the lagrangian multipliers are not updated at each execution of this step. This approach is motivated by the intuitive consideration that the computations involved in determining optimal multipliers are too time consuming and may not justify the improvement obtained in the lower bound value. We thus prefer to sacrifice the sharpness of the bound to achieve the speed of executing

this step. This approach is also found to be highly conducive to reoptimisation from one subproblem to the other. Since v_i^k, once computed initially, are fixed throughout, the (LR_2) reduces to

$$Z_2 = \min \sum_{j\varepsilon J} \sum_{i\varepsilon I} C_{ij}\, x_{ij} - \sum_{i\varepsilon I} \sum_{k\varepsilon K} v_i^k\, b_i^k$$

subject to (1.1) and (1.3)

where,

$$C_{ij} = c_{ij} + \sum_{k\varepsilon K} v_i^k\, a_{ij}^k$$

The relaxed subproblem to be solved is

$$\text{Min} \sum_{i\varepsilon I} \sum_{j\varepsilon J} c_{ij}\, y_{ij}^l + \sum_{i\varepsilon I} \sum_{j\varepsilon J} C_{ij}\, x_{ij} - \sum_{i\varepsilon I} \sum_{k\varepsilon K} v_i^k\, \bar{b}_i^k$$

subject to (1.1) and (1.3)

where,

$$\bar{b}_i^k = b_i^k - \sum_{j\varepsilon J} a_{ij}^k\, y_{ij}^l, \quad \forall\ i\varepsilon I, \ \forall\ k\varepsilon K$$

Clearly, the lower bound for the subproblem is

$$LB = \sum_{i\varepsilon I} \sum_{j\varepsilon J} c_{ij}\, y_{ij}^l - \sum_{i\varepsilon I} \sum_{k\varepsilon K} v_i^k\, \bar{b}_i^k + \sum_{j\varepsilon J,\ \sum_{i\varepsilon I} y_{ij}^l = 0} \left[\min_{\substack{i\varepsilon I \text{ and } y_{ij}^o \neq 1 \\ \text{and } (i,j) \notin S}} \{C_{ij}\} \right]$$

where S is the set of variables whose $a_{ij}^k > \bar{b}_i^k$ for any k.

Now, from the above expression it is clear that when a variable is fixed to one or zero, the computations needed to

update the terms in the expression are very less. For example, when a variable $x_{\alpha\beta}$ is fixed to one, the second term in the expression is updated since $\bar{b}_i^k$ changes. Then it is checked whether there is any task j, which is assigned to agent α previously, whose $a_{\alpha j}^k$ value is greater than the new $\bar{b}_\alpha^k$ value for any k. If there are such tasks they will be reallocated and the lower bound will be updated accordingly. This needs computations of the order of qm, where q is the number of tasks to be reallocated. Otherwise the lower bound will not change and so it need not be recalculated, which is the case when a feasible solution is found. Similarly, when a variable $x_{\alpha\beta}$ is fixed to zero, the second term is updated and the task will be reallocated and the third term in the expression is updated accordingly. Also, we check whether any tasks have to be reallocated to this agent because its $\bar{b}_\alpha^k$ value has increased when we fixed the variable $x_{\alpha\beta}$ to zero. Thus, the lower bound need not be calculated afresh at each node, but it needs to be updated only.

Step 5: After solving (P^h), we will try to prune the subproblem that is, to remove the subproblem from further consideration. This pruning can be divided into four steps. In the first step, we make a check whether Z^h, the lower bound, is greater than Z, the incumbent solution value at that instant of the branch and bound tree. If it is so, the subproblem is pruned and it is removed from further consideration. However,

LIBRARY
17588

if this condition is not satisfied, we proceed to the second step. In this step, we check whether the solution x(v) is feasible to (P), that is, whether the solution x(v) satisfies the resource constraints of (P) also. The feasibility check can be made easily by checking for a strictly positive component in S^v. If all $S^v \leq 0$ does not hold, the pruning process ends without pruning the subproblem. If all $S^v \leq 0$, we proceed to the third step, where we check whether the solution of (P^h) is better than the incumbent solution. If it is so, the incumbent solution is updated. Whatever is the result in the third step, we proceed to the fourth step. Here, we will make another attempt to fathom the subproblem. When x(v) is feasible to problem (P) also, the following condition will be satisfied.

$$Z^h = cx(v) + vS^v \leq Z_{opt} \leq cx(v)$$

where the last inequality follows from the verifications in the second step of the pruning process, of the feasibility of x(v). Also all $S^v \leq 0$, because x(v) is feasible to (P). Hence, if $vS^v = 0$, then

$$Z^h = Z_{opt} = cx(v)$$

so that v and x(v) are optimal solutions to (PM) and the subproblem (P^h), respectively. Note that, if subproblem (P^h) is fathomed in the fourth step of pruning process, then x(v) will always have become the new incumbent, because

$$cx(v) = Z^h \leq Z$$

where the first equality follows because $Z^h = Z_{opt} = cx(v)$ and the second inequality follows because the subproblem was not fathomed in the first step of the pruning process.

Step 6: After the pruning attempt failed, (P^h) must be partitioned into two subproblems. For this, we have to choose a separation variable $x_{\alpha\beta}$ to partition the solution set of (P^h) into two subsets. There are several strategies to select α, β and a few are explained below.

In the strategy which we adopted in our thesis, α, β are selected as follows. α is the one among the set of agents for which

$$T_\alpha = \max_i \left[\sum_{k=1}^{p} \max \left[0, \sum_{j \in J} a_{ij}^k x_{ij} - b_i^k \right] \right]$$

In other words, we are selecting the agent with maximum total exceeded capacity.

Then β is selected from among the tasks that are assigned to α and which has the maximum penalty p_β. The penalty p_j is defined as the excess cost that will be incurred when the task j is shifted from the present agent to the agent with the next highest cost.

The variable chosen by this rule represents a task j that is best kept with agent i considering both the penalty for switching the tasks and the resources available to the agent.

There are other strategies for selecting the separation variable. One such strategy is to select that agent whose resource has exceeded by a maximum amount than any other resource of any agent. Then selecting the task is same as the above strategy, that is, the highest penalty task. This strategy also worked well though not as good as the first strategy. This is because an agent, all resources of which has exceeded the capacity, is ignored.

3.3 NUMERICAL EXAMPLE:

In this section we give a small numerical example and the branch and bound tree for the same. The objective function values (c_{ij}) and the coefficients (a^k_{ij}) of the multiple assignment constraints for a problem with three agents, five tasks and two resources are shown in Tables 1 and 2 respectively. In this example, the right hand side values (b^k_i) of the multiple assignment constraints are 38 for all $i \in I$ and $k \in K$.

Initially (LR_2) is solved to obtain the solution $x_{31} = 1$, $x_{12} = 1$, $x_{13} = 1$, $x_{24} = 1$, $x_{25} = 1$ and all other $x_{ij} = 0$. This solution is not feasible to (P). The lower bound (LB) provided by this solution is 100.

The variable x_{24} is selected as the separation variable and the candidate problem with $x_{24} = 1$ is solved next because there are no other tasks which are assigned to agent 2 and which can be fixed to the same agent. The solution to this candidate

problem is $x_{31} = 1$, $x_{12} = 1$, $x_{13} = 1$, $x_{24} = 1$, $x_{35} = 1$ and all other $x_{ij} = 0$, which is again not feasible to (P). The LB is 102.

Now, variable x_{35} is selected for branching and the candidate problem is solved to obtain the solution $x_{11} = 1$, $x_{12} = 1$, $x_{13} = 1$, $x_{24} = 1$, $x_{35} = 1$ and all other $x_{ij} = 0$ with LB = 106.

Now, the variable x_{11} and then x_{12} are selected as separation variables and are fixed to one and the next candidate problem with $x_{24} = 1$, $x_{35} = 1$, $x_{11} = 1$, $x_{12} = 1$, is solved at node (5) to obtain the solution $x_{11} = 1$, $x_{12} = 1$, $x_{33} = 1$, $x_{24} = 1$, $x_{35} = 1$ and all other $x_{ij} = 0$. This solution is feasible to (P)

Table 1: Objective function coefficients

		Tasks				
		1	2	3	4	5
	1	22	16	37	45	34
Agents	2	32	19	47	22	15
	3	11	42	34	43	15

Table 2: Multiple assignment coefficients.

		k=1					k=2				
		Tasks					Tasks				
		1	2	3	4	5	1	2	3	4	5
	1	13	6	10	6	7	8	18	15	18	22
Agents	2	19	13	6	7	12	12	21	15	23	21
	3	17	24	21	21	10	25	15	16	19	17

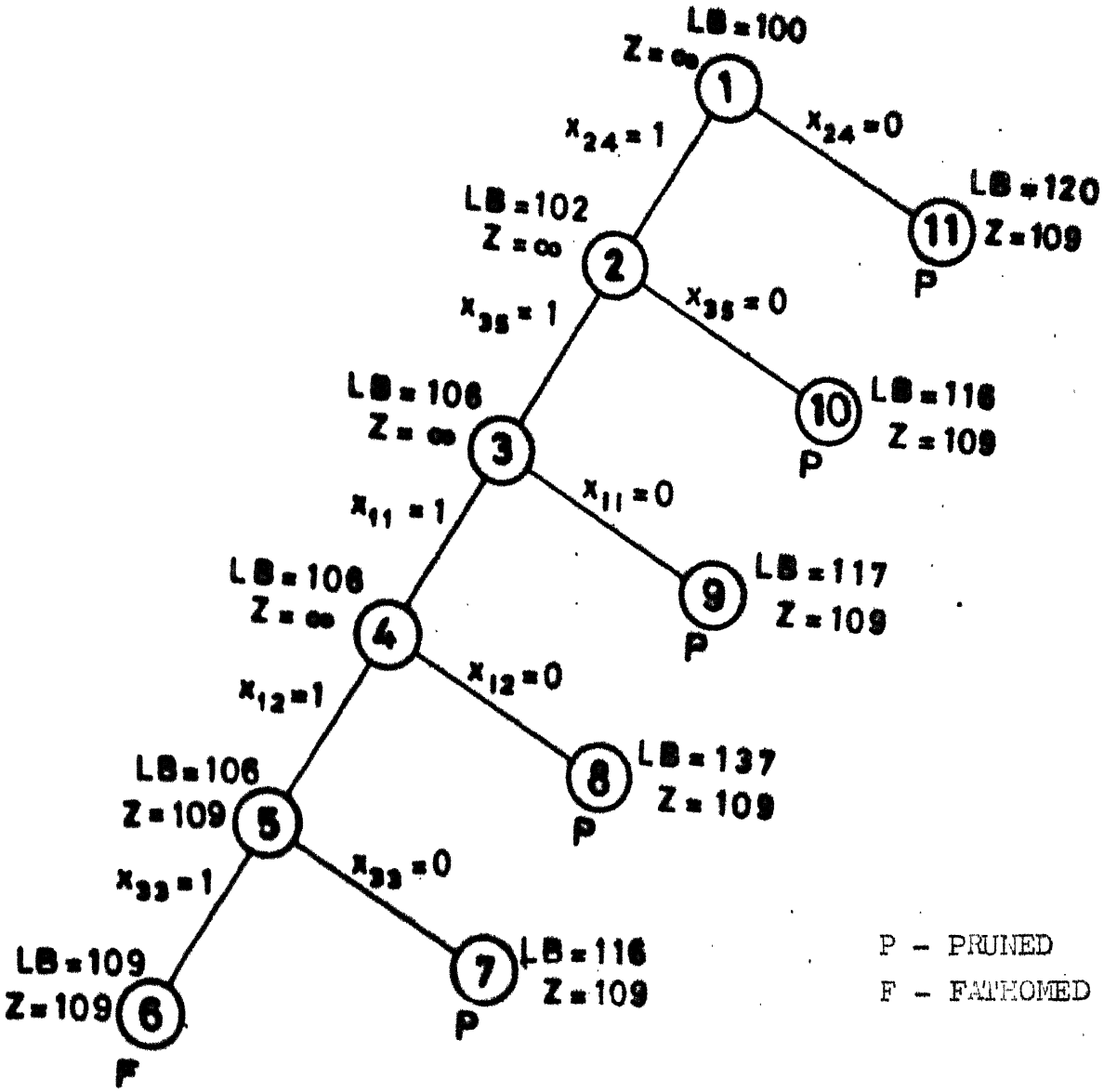

LB = 100
Z = ∞
1
$x_{24} = 1$
$x_{24} = 0$
LB = 102
Z = ∞
2
11
LB = 120
Z = 109
P
$x_{35} = 1$
$x_{35} = 0$
LB = 106
Z = ∞
3
10
LB = 116
Z = 109
P
$x_{11} = 1$
$x_{11} = 0$
LB = 108
Z = ∞
4
9
LB = 117
Z = 109
P
$x_{12} = 1$
$x_{12} = 0$
LB = 106
Z = 109
5
8
LB = 137
Z = 109
P
$x_{33} = 1$
$x_{33} = 0$
LB = 109
Z = 109
6
F
7
LB = 116
Z = 109
P
P - PRUNED
F - FATHOMED

FIGURE 2

with an objective function value of 109 and the incumbent is updated accordingly. Now, we fix all the remaining tasks which are not fixed in the previous candidate problem and back track. While back tracking the candidate problem with $x_{33} = 0$ and all the remaining tasks fixed is considered next. This gives a lower bound of 115 and is pruned. The process is repeated until all the nodes are pruned and finally the optimum solution is $x_{11} = 1$, $x_{12} = 1$, $x_{33} = 1$, $x_{24} = 1$, $x_{35} = 1$ and all other $x_{ij} = 0$. The solution value is 109. In this case, the first feasible solution is also the optimum solution.

The complete branch and bound tree for this example is given in Fig. 2. The numbers in the circles indicate the node numbers, LB designates the lower bound on the node and Z is the incumbent value.

3.4 COMPUTATIONAL RESULTS:

In this section we report the computational performance of the branch and bound algorithm for MRGAP in solving problems of different sizes. We also report the performance of the ε - optimal algorithm with $\varepsilon = 0.01$.

The branch and bound algorithm was coded in FORTRAN IV and tested on DEC 10 computer system under multi programming environment. The program was executed on randomly generated problems with varying number of agents (m), tasks (n) and resources (p). We considered six combinations of tasks and

agents and for each combination three problems were solved with the number of resources equal to 2, 4 and 6. The problems were generated using the same scheme as used by Ross and Soland to generate generalized assignment problems. The values of c_{ij} and a_{ij}^k were generated from a uniform probability distribution with intervals as [10,50] and [5, 25] respectively. To establish binding multiple assignment constraints, each b_i^k was set equal to $0.6 \times \frac{n}{m} \times 15 + 0.4 \times R$, where

$$R = \max_{i \in I} \left[\sum_{j \in J} a_{ij}^k x_{ij} \right]$$

and x_{ij} are the solution to the initial relaxation of (P). Ross and Soland reported that such a scheme attempts to set tight constraints and avoids the possibility of generating trivial problems which might be solved in the initial relaxation. In general, based on the range of values of the a_{ij}^k, one would expect random problems to be infeasible if each $b_i^k < \frac{n}{m} \times 15$ and trivial if each $b_i^k \geq R$.

For each problem solved we noted the number of nodes enumerated in the branch and bound tree and the computational time. The times given in the tables were measured by real time clock, accurate upto one millisecond, and does not include input and output times. We also noted the number of nodes examined and the computational time taken and the objective function value for the first feasible solution. All these details are given in Table 3. The performance of the ε-optimal

algorithm (the branch and bound algorithm in which a node is pruned whenever LB x $(1 + \varepsilon) \geq$ incumbent value) is given in Table 4. The following conclusions can be drawn from the tables.

1. The algorithm can solve problems upto 1000 variables and 6 resources within reasonable computational effort. The computational times are found to be more sensitive to the number of agents than the number of tasks. Further, the computational times do not appear to grow exponentially with increase in the number of resources. In fact, increasing the number of resources reduces the number of feasible solutions and, thereby, reducing the amount of implicit search.

2. The algorithm examines a fairly large number of nodes in the branch and bound tree before terminating. This is because the lower bound provided by the relaxation (LR_2) is somewhat loose. However, the bound is obtained with little computational effort which makes up the looseness of the bound.

3. The algorithm obtains a very good feasible solution, which is often optimum, early in the implicit search. The optimum solution is also invariably obtained in the early stages and rest of the time is spent in proving the optimality of the solution. Hence, premature termination of the algorithm can be recommended for large sized problems.

4. The performance of the ε - optimal algorithm is also very encouraging. The algorithm is able to solve problems upto 8000 variables and 6 resources within 1 percent of the optimality in less than 3 minutes.

3.5 CONCLUDING REMARKS:

In Chapter I, we observed that MRGAP has useful practical applications. We gave the statement of the problem and the applications.

In Chapter II, we reviewed the literature available for the assignment models and the lagrangian relaxation concepts we also gave a brief account of concepts of lagrangian relaxation and subgradient optimisation.

We developed a branch and bound algorithm in Chapter III with special features like calculation of lagrangian multipliers only at the root node and calculation of lower bound only at intermediate nodes etc.

Finally, we have done extensive computational study on the algorithm. We observed that the algorithm is performing quite well. It was able to solve 1000 0-1 variables problem with 6 resources in about 60 seconds of CPU time. We also observed that the ε - optimal algorithm with ε = 0.01 was very effective in solving large sized problems, and it was able to solve a 8000 0-1 variables problem with 6 resources in about 3 minutes.

Table 3: Performance of the exact branch and bound algorithm (computational times are in seconds on DEC 10).

Problem No.	m	n	p	First Feasible Solution			Optimum Solution	
				$\bar{Z}/Z$	Nodes Examined	Computational Time	Nodes Examined	Computational Time
1	5	100	2	1.000	0	0.39	495	0.83
2			4	1.001	44	0.77	2179	5.30
3			6	1.012	80	1.09	8042	13.86
4	10	50	2	1.000	25	0.44	1258	2.61
5			4	1.022	37	0.75	176	1.18
6			6	1.000	37	1.06	2005	3.86
7	10	100	2	1.001	11	0.84	113	1.15
8			4	1.000	30	1.46	160	1.97
9			6	1.000	99	2.09	135	2.68
10	15	30	2	1.009	23	0.57	384	1.33
11			4	1.040	22	0.99	1210	3.84
12			6	1.000	9	1.37	3001	8.39
13	15	75	2	1.000	26	1.37	11270	20.04
14			4	1.000	12	2.31	75	2.62
15			6	1.001	23	3.32	165	4.08
16	20	50	2	1.003	16	1.59	4785	8.21
17			4	1.000	36	2.74	500	3.57
18			6	1.000	18	3.86	13925	54.72

Table 4: Performance of the ε - optimal algorithm with $\varepsilon = 0.01$.
(Computational times are in seconds on DEC 10).

Problem No.	m	n	p	Nodes Examined	Computational Time
1	10	100	2	22	1.14
2			4	60	1.90
3			6	104	2.65
4	10	200	2	200	1.98
5			4	124	4.42
6			6	174	6.08
7	20	100	2	190	3.47
8			4	132	5.89
9			6	880	8.30
10	20	200	2	112	3.25
11			4	194	12.36
12			6	200	17.45
13	40	200	2	76	8.36
14			4	52	45.99
15			6	88	175.67

REFERENCES

1. Balachandran, V., An integer generalized transportation model for optimal job assignment in computer networks, Operations Research 24, 742-759 (1976).

2. Barr, R., Glover, F., Klingman, D., The alternating basis algorithm for assignment problems, Mathematical Programming 3, 1-13 (1977).

3. Fisher, M.L., The lagrangian relaxation method for solving integer programming problems, Management Science, 27, 1-17 (1981).

4. Fisher, M.L., Jaikumar, R., and Van Wassenhove, L.N., A multiplier adjustment method for the generalized assignment problem, Decision Sciences Working Paper, Univ. of Pennsylvania (1980).

5. Gavish, B., Topological design of centralised computer networks, Networks 12, 355-378 (1982).

6. Held, M., and Karp, R.M., The travelling salesman problem and minimum spanning trees, Operations Research, 18, 1138-1162 (1970).

7. Held., M., Wolfe, P., and Crowder, H.P., Validation of subgradient optimisation, Mathematical Programming 6, 62-88 (1974).

8. Hung, M.S., and Rom, W.O., Solving the assignment problem by relaxation, Operations Research 28, 969-982 (1980).

9. Klastorin, T.D., An effective subgradient algorithm for the generalized assignment problem, Computers and Operations Research 6, 155-164 (1979).

10. Kuhn, H.W., The Hungarian method for the assignment problem, Naval Research Logistics Quarterly 3, 253-258 (1956).

11. Ross, G.T., and Soland, R.M., A branch and bound algorithm for the generalized assignment problem, Mathematical Programming 8, 91-103 (1975).

12. Ross, G.T., and Zoltners, A.A., Weighted assignment models and their applications, Management Science 25, 683-696 (1979).

13. Shogan, A.W., Constructing a minimal cost spanning tree subject to resource constraints and flow requirements, Networks 13, 169-190 (1983).

14. Syslo, M.M., Narsingh Deo, Kowalik, J.S., Combinatorial Algorithms, Prentice Hall Inc., Englewood Cliffs, NJ 07632.

```
C**********************************************************************
C**   THIS PROGRAM GENERATES AND SOLVES MULTI-RESOURCE GENERALISED
C**   ASSIGNMENT PROBLEMS.
C**********************************************************************
C**   THE INPUT PARAMETERS ARE EXPLAINED BELOW
C**   M ::-- NUMBER OF AGENTS
C**   N ::-- NUMBER OF TASKS
C**   P ::-- NUMBER OF RESOURCES
C**   ISEED::-- SEED
C
C**   IMPORTANT VARIABLES IN THE PROGRAM ARE EXPLAINED BELOW
C**   IZOO ::-- INCUMBENT SOLUTION VALUE
C**   IX   ::-- INCUMBENT SOLUTION
C**   A    ::-- MULTIPLE ASSIGNMENT CONSTRAINT COEFFICIENTS
C**   C    ::-- OBJECTIVE FUNCTION COEFFICIENTS
C**   B    ::-- CAPACITY MATRIX
C**   PB   ::-- CAPACITY MATRIX OF THE SUBPROBLEM AT ANY INSTANT
C**   X(I,J) = 1  IF TASK J IS ASSIGNED TO AGENT I
C**          = 0  OTHERWISE
C**   TE   ::-- SUM OF A(I,J,K)*X(I,J) FOR ALL I AND K IN THE
C**               INITIAL CALCULATIONS AT THE ROOT NODE
C**        ::-- A(I,J,K)*X(I,J)-PB(I,K) THERE AFTER
C**   V    ::-- LAGRANGIAN MULTIPLIER MATRIX
C**   CNEW ::-- SUM OF A(I,J,K)*V(I,K) FOR ALL I AND J
C**   SE   ::-- PENALTY ARRAY
C**   KX   ::-- HEAP TO ORDER THE TASKS IN THE DECREASING ORDER OF
C**               PENALTY
C**   FL(I,J) = TRUE  IF VARIABLE X(I,J) IS FIXED TO ZERO
C**               = FALSE  OTHERWISE
C**   FLAG(J) = TRUE  IF TASK J IS FIXED TO SOME AGENT
C**           = FALSE  OTHERWISE
C**   IS   ::-- STACK OF AGENT INDICES
C**   JS   ::-- STACK OF TASK INDICES
C**   IVAL ::-- POINTER TO INDICATE THE SUBPROBLEM STATUS
C**   CONE(J)::-- AGENT TO WHICH TASK J IS ASSIGNED
C**********************************************************************
      COMMON M,N,P,A(25,200,6),C(25,200),B(25,6),IX(200),IZOO
      INTEGER P
      REAL LARGE,IZOO,MAX
      READ(21,*)M,N,P,ISEED
      CALL GENRAT(M,N,P,A,C,B,ISEED)
      CALL MRGAP
      IIZOO=IZOO
      WRITE(23,100)IIZOO
```

```
0046          MPK=1
0047          MP=20
0048          IN=N
0049    110   IF(IN.GT.20)MP=MPK+IN-1
0050          WRITE(23,101)(I,I=MPK,MP)
0051    101   FORMAT(/1X,'JOBS',3X,20(I4,1X))
0052          WRITE(23,103)
0053    103   FORMAT(/,8X,20(1X,'---',1X))
0054          WRITE(23,104)
0055    104   FORMAT(8X,20('|',3X,'|'))
0056          WRITE(23,105)(IX(I),I=MPK,MP)
0057    105   FORMAT(1X,'AGENTS',1X,20('|',I3,'|'))
0058          WRITE(23,106)
0059    106   FORMAT(8X,20(1X,'---',1X)/)
0060          IN=IN-20
0061          MPK=MP+1
0062          MP=MP+20
0063          IF(IN.GT.0)GO TO 110
0064          STOP;END
0065          SUBROUTINE MRGAP
0066    **    ************************************************************
0067    **    THIS SUBROUTINE TAKES THE PROBLEM DATA AND GIVES THE
0068    **    OPTIMUM SOLUTION TO A M-R-G-A PROBLEM
0069    **    ************************************************************
0070          COMMON M,N,P,A(25,200,6),C(25,200),B(25,6),IX(200),IZOO
0071          DIMENSION PB(25,6),V(25,6),TE(25,6)
0072          DIMENSION KX(200),SE(200),IS(1000),JS(1000),IVAL(1000)
0073          DIMENSION CNEW(25,200)
0074          INTEGER X(25,200),P,TOP,FIRST,CONE(200)
0075          LOGICAL FL(25,200),FLAG(200),FEASE,ITRACK
0076          REAL KLB,LB,IZOO,ICOST,LARGE,MAX
0077          LIM=2
0078          EPS1=0.00001
0079          EPS2=0.00001
0080          LIMITR=M
0081          IF(M.GE.6)LIMITR=2*M
0082          ICC=0
0083          DO 7 I=1,M
0084          DO 7 K=1,P
0085    7     PB(I,K)=B(I,K)
0086          LARGE=10000.
0087          IZOO=0.
0088          PMIN=10000.
0089          DO 9 I=1,M
```

```
00900           DO 9 K=1,P
00910   9       V(I,K)=0.0
00920   **      ****** INITIALISE IZOD ******
00930           DO 22 J=1,N
00940           MAX=0.
00950           DO 21 I=1,M
00960           X(I,J)=0
00970           IF(C(I,J).LE.MAX)GO TO 21
00980           MAX=C(I,J)
00990           INDM=I
01000   21      CONTINUE
01010           IZOD=IZOD+C(INDM,J)
01020   22      CONTINUE
01030
01040           DO 31 J=1,N
01050           FLAG(J)=.FALSE.
01060           CONE(J)=0.
01070           DO 31 I=1,M
01080   31      FL(I,J)=.FALSE.
01090           TOP=0
01100           FIXVAL=0.
01110           ICOU=0
01120           ITOTAL=0
01130           CALL RTIME(N1)
01140           PLAMDA=0.2;ICOUNT=1;PREVLB=0.;ITER=0
01150
01160   **      **** CALCULATION OF LAGRANGE MULTIPLIERS AT THE ROOT NODE ****
01170   11      ITER=ITER+1;FEASE=.TRUE.
01180           ICOST=0.
01190           LB=0.
01200           ELEMEN=0.
01210           DO 40 I=1,M
01220           DO 40 K=1,P
01230           PE(I,K)=0.
01240   40      ELEMEN=ELEMEN+V(I,K)*PB(I,K)
01250           ITRACK=.FALSE.
01260           DO 30 J=1,N
01270           PMIN=LARGE
01280           DO 20 I=1,M
01290           TEMP=0.
01300           CNEW(I,J)=LARGE
01310           DO 10 K=1,P
01320           IF(A(I,J,K).GT.PB(I,K)) GO TO 20
01330   10      TEMP=V(I,K)*A(I,J,K)+TEMP
```

```
0135@           TEMP=TEMP+C(I,J)
0136@           IF(TEMP.GE.PMIN) GO TO 20
0137@           PMIN=TEMP
0138@           INDEX=I
0139@    20     CONTINUE
0140@           IF(PMIN.GE.LARGE) ITRACK=.TRUE.
0141@    79     CONE(J)=INDEX
0142@           ICOST=ICOST+C(INDEX,J)
0143@           LB=LB+PMIN
0144@           DO 25 K=1,P
0145@    26     TE(INDEX,K)=TE(INDEX,K)+A(INDEX,J,K)
0146@    30     CONTINUE
0147@           IF(ITRACK)GO TO 16
0148@           KLB=LB
0149@           LB=KLB-ELEMEN
0150@           IF(PREVLB.GE.(1.001*LB))GO TO 45
0151@           ICOUNT=1
0152@           PREVLB=LB
0153@    45     SUM=0.
0154@           PMAX=-10000.
0155@           DO 50 I=1,M
0156@           DO 50 K=1,P
0157@           TE(I,K)=TE(I,K)-PB(I,K)
0158@           XX=TE(I,K)
0159@           SUM=SUM+XX*XX
0160@           IF(XX.GT.0)FEASE=.FALSE.
0161@    50     CONTINUE
0162@           IF(FEASE) GO TO 49
0163@           IF(ITER.GT.LIMITR)GO TO 49
0164@           TL=PLAMDA*(IZOD-PREVLB)/SUM
0165@           DO 60 I=1,M
0166@           DO 60 K=1,P
0167@           V(I,K)=V(I,K)+TL*TE(I,K)
0168@    60     IF(V(I,K).LT.0) V(I,K)=0.
0169@           ICOUNT=ICOUNT+1
0170@           IF(ICOUNT.LE.LIM)GO TO 70
0171@           PLAMDA=PLAMDA/2.
0172@    70     IF(PLAMDA.LT.EPS1)GO TO 49
0173@           IF(TL.LT.EPS2) GO TO 49
0174@           GO TO 11
0175@    49     DO 85 J=1,N
0176@           ICONJ=CONE(J)
0177@           X(ICONJ,J)=1
0178@           SECOND=LARGE
```

```
01790           DO 85 I=1,M
01800           IF(ICONJ.EQ.I)GO TO 86
01810           TEMP=CNEW(I,J)+C(I,J)
01820           IF(TEMP.GT.SECOND)GO TO 86
01830           SECOND=TEMP
01840   86      CONTINUE
01850           SE(J)=SECOND-CNEW(ICONJ,J)-C(ICONJ,J)
01860   85      CONTINUE
01870           PMAX=0.
01880           DO 188 I=1,M
01890           TEMP=0.
01900           DO 88 K=1,P
01910           IF(TE(I,K).LE.0)GO TO 88
01920           TEMP=TEMP+TE(I,K)
01930   88      CONTINUE
01940           IF(TEMP.LE.PMAX)GO TO 188
01950           PMAX=TEMP
01960           INDI=I
01970   188     CONTINUE
01980
01990   **      ** CHECKS THE FEASIBILITY OF THE SOLUTION AND UPDATES THE
02000   **      INCUMBENT IF NECESSARY **
02010
02020   100     IF(.NOT.FEASE)GO TO 17
02030           ICH=ICH+1
02040           IF((ICOST+FIXVAL).GE.IZOD) GO TO 19
02050           IZOD=ICOST+FIXVAL
02060           IF(ICH.GT.1)GO TO 101
02070           CALL RTIME(N3)
02080           IFEAS=N3-N1
02090           WRITE(23,1110)IZOD
02100   1110    FORMAT(1X,'THE FIRST FEASIBLE SOLUTION VALUE IS ',F8.2)
02110           WRITE(23,1111)IFEAS
02120   1111    FORMAT(1X,'TIME TAKEN FOR THE FIRST FEASIBLE SOLUTION  ',I6,'
02130           1  MILLI SECONDS')
02140           WRITE(23,1112)ICOU
02150   1112    FORMAT(1X,' NO. OF NODES EXAMINED FOR THE FIRST FEASIBLE
02160           1 SOLUTION ',I6)
02170   101     DO 14 J=1,N
02180   14      IX(J)=CONE(J)
02190           IF(LBL.GE.IZOD)GO TO 16
02200   19      KXOJJ=0
02210           DO 180 J=1,N
02220           IF(FLAG(J))GO TO 180
```

```
02240           KX(KXCOU)=J
02250   180     CONTINUE
02260
02270   **      ORDERS THE JOBS IN DECREASING ORDER OF PENALTY USING HEAP SORT
02280           IF(KXCOU.LE.1)GO TO 181
02290           KKX=KXCOU
02300           IFIRST=KXCOU/2
02310   219     FIRST=IFIRST
02320   310     J=FIRST
02330           ITEMP=KX(J);TEMP=SE(ITEMP)
02340   320     K=2*J
02350           IF(K.GT.KKX)GO TO 330
02360           IF((K.LT.KKX).AND.(SE(KX(K+1)).LT.(SE(KX(K)))))K=K+1
02370           IF(SE(KX(K)).GT.TEMP)GO TO 330
02380           KX(J)=KX(K)
02390           J=K
02400           GO TO 320
02410   330     KX(J)=ITEMP
02420           FIRST=FIRST-1
02430           IF(FIRST.GT.0)GO TO 310
02440           LTEMP=KX(1)
02450           KX(1)=KX(KKX)
02460           KX(KKX)=LTEMP
02470           KKX=KKX-1
02480           IF(KKX.EQ.1)GO TO 181
02490           IFIRST=1
02500           GO TO 219
02510
02520   **      FIXES THE JOBS IN THE DECREASING ORDER OF PENALTY
02530   181     DO 18 KK=1,KXCOU
02540           J=KX(KK)
02550           IF(FLAG(J)) GO TO 18
02560           TOP=TOP+1
02570           ICOU=ICOU+1
02580           FLAG(J)=.TRUE.
02590           ICONJ=CONE(J)
02600           IS(TOP)=ICONJ
02610           JS(TOP)=J
02620           IVAL(TOP)=1
02630           DO 71 K=1,P
02640    71     PB(ICONJ,K)=PB(ICONJ,K)-A(ICONJ,J,K)
02650           ELEMEN=ELEMEN-CNEW(ICONJ,J)
02660           KLB=KLB-C(ICONJ,J)-CNEW(ICONJ,J)
02670           FIXVAL=FIXVAL+C(ICONJ,J)
```

```
02680    18      CONTINUE
02690            ICOST=0.0
02700            GO TO 16
02710
02720    **      FIXES THE JOBS OF THE AGENT SELECTED FOR BRANCHING
02730    17      KXCOU=0
02740            DO 23 J=1,N
02750            IF(X(INDI,J).EQ.0)GO TO 23
02760            IF(FLAG(J))GO TO 23
02770            KXCOU=KXCOU+1
02780            KX(KXCOU)=J
02790    23      CONTINUE
02800            KKX=KXCOU
02810            IF(KKX.LE.1)GO TO 35
02820            IFIRST=KXCOU/2
02830    119     FIRST=IFIRST
02840    210     J=FIRST
02850            ITEMP=KX(J);TEMP=SE(ITEMP)
02860    220     K=2*J
02870            IF(K.GT.KKX)GO TO 230
02880            IF((K.LT.KKX).AND.(SE(KX(K+1)).LT.(SE(KX(K)))))K=K+1
02890            IF(SE(KX(K)).GT.TEMP)GO TO 230
02900            KX(J)=KX(K)
02910            J=K
02920            GO TO 220
02930    230     KX(J)=ITEMP
02940            FIRST=FIRST-1
02950            IF(FIRST.GT.0)GO TO 210
02960            LTEMP=KX(1)
02970            KX(1)=KX(KKX)
02980            KX(KKX)=LTEMP
02990            KKX=KKX-1
03000            IF(KKX.EQ.1)GO TO 35
03010            IFIRST=1
03020            GO TO 119
03030    35      ISTAR=INDI
03040            DO 28 J=1,KXCOU
03050            IF(FLAG(KX(J)))GO TO 28
03060            DO 29 K=1,P
03070    29      IF(A(INDI,KX(J),K).GT.PB(INDI,K))GO TO 28
03080            JSTAR=KX(J)
03090            TOP=TOP+1
03100            ICOU=ICOU+1
03110            IS(TOP)=ISTAR
```

```
03130           IVAL(TOP)=1
03140           X(ISTAR,JSTAR)=1
03150           CONE(JSTAR)=ISTAR
03160           FLAG(JSTAR)=.TRUE.
03170           DO 24 K=1,P
03180    24     PB(ISTAR,K)=PB(ISTAR,K)-A(ISTAR,JSTAR,K)
03190           FIXVAL=FIXVAL+C(ISTAR,JSTAR)
03200           KLB=KLB-C(ISTAR,JSTAR)-CNEW(ISTAR,JSTAR)
03210           ELEMEN=ELEMEN-CNEW(ISTAR,JSTAR)
03220           ICOST=ICOST-C(ISTAR,JSTAR)
03230    28     CONTINUE
03240
03250    **     CHECKS THE JOBS TO BE REALLOCATED
03260    76     ITRACK=.FALSE.
03270           DO 77 KK=1,KXCOU
03280           J=KK(KK)
03290           IF (FLAG(J)) GO TO 77
03300           PMIN=LARGE
03310           SECOND=LARGE
03320           DO 48 I=1,M
03330           IF(I.EQ.INDI)GO TO 48
03340           IF (FL(I,J)) GO TO 48
03350           DO 41 K=1,P
03360    41     IF (A(I,J,K).GT.PB(I,K)) GO TO 48
03370           TEMP=C(I,J)+CNEW(I,J)
03380           IF(TEMP.GE.PMIN) GO TO 43
03390           SECOND=PMIN
03400           PMIN=TEMP
03410           INDEX=I
03420           GO TO 48
03430    43     IF(TEMP.GE.SECOND)GO TO 48
03440           SECOND=TEMP
03450    48     CONTINUE
03460           IF(PMIN.GE.LARGE) ITRACK=.TRUE.
03470           X(INDI,J)=0
03480           X(INDEX,J)=1
03490           SE(J)=SECOND-PMIN
03500           CONE(J)=INDEX
03510           ICOST=ICOST-C(INDI,J)+C(INDEX,J)
03520           DO 44 K=1,P
03530           TE(INDEX,K)=TE(INDEX,K)+A(INDEX,J,K)
03540    44     TE(INDI,K)=TE(INDI,K)-A(INDI,J,K)
03550           TEMP=C(INDI,J)+CNEW(INDI,J)
03560           KLB=KLB-TEMP+PMIN
```

```
03570   77      CONTINUE
03580           IF(ITRACK)GO TO 16
03590
03600   **      CHECK FOR PRUNING AND SELECTS THE AGENT FOR BRANCHING
03610   93      FEASE=.TRUE.
03620           LB=KLB-ELEMEN+FIXVAL
03630           LB=LB+0.99999
03640           IF(LB.GE.IZOD) GO TO 16
03650           PMAX=-10000.0
03660           DO 51 I=1,M
03670           DO 51 K=1,P
03680           IF(TE(I,K).GT.0)FEASE=.FALSE.
03690   51      CONTINUE
03700           IF(FEASE)GO TO 100
03710           DO 1151 I=1,M
03720           TEMP=0.
03730           DO 151 K=1,P
03740           IF(TE(I,K).LE.0)GO TO 151
03750           TEMP=TEMP+TE(I,K)
03760   151     CONTINUE
03770           IF(TEMP.LE.PMAX)GO TO 1151
03780           PMAX=TEMP
03790           INDI=I
03800   1151    CONTINUE
03810           GO TO 100
03820
03830   **      BACK TRACK
03840   16      IF(TOP.EQ.0)GO TO 78
03850           KK=IVAL(TOP)
03860           I=IS(TOP)
03870           J=JS(TOP)
03880           IF(KK.EQ.2)GO TO 130
03890           IVAL(TOP)=2
03900           FL(I,J)=.TRUE.
03910           FLAG(J)=.FALSE.
03920           FIXVAL=FIXVAL-C(I,J)
03930           ICOST=ICOST+C(I,J)
03940           DO 120 K=1,P
03950   120     PB(I,K)=PB(I,K)+A(I,J,K)
03960           ELEMEN=ELEMEN+CNEW(I,J)
03970           PMIN=LARGE
03980           SECOND=LARGE
03990           DO 81 II=1,M
04000           IF(FL(II,J))GO TO 81
```

```
04020    82      IF(A(II,J,K).GT.PB(II,K))GO TO 81
04030            TEMP=C(II,J)+CNEW(II,J)
04040            IF(TEMP.GE.PMIN)GO TO 83
04050            SECOND=PMIN
04060            PMIN=TEMP
04070            INDEX=II
04080            GO TO 81
04090    83      IF(TEMP.GE.SECOND)GO TO 81
04100            SECOND=TEMP
04110    81      CONTINUE
04120            IF(PMIN.GE.LARGE) GO TO 16
04130            ICONJ=CONE(J)
04140            X(ICONJ,J)=0
04150            X(INDEX,J)=1
04160            SE(J)=SECOND-PMIN
04170    84      KLB=KLB+PMIN
04180            ICOST=ICOST+C(INDEX,J)-C(ICONJ,J)
04190            DO 108 K=1,P
04200            TE(ICONJ,K)=TE(ICONJ,K)-A(ICONJ,J,K)
04210    108     TE(INDEX,K)=TE(INDEX,K)+A(INDEX,J,K)
04220            CONE(J)=INDEX
04230            DO 52 J=1,N
04240            IF(FLAG(J)) GO TO 52
04250            IF(FL(I,J)) GO TO 52
04260            IF(CONE(J).EQ.I) GO TO 52
04270            DO 54 K=1,P
04280    54      IF(A(I,J,K).GT.PB(I,K)) GO TO 52
04290            ICONJ=CONE(J)
04300            TEMP=C(I,J)+CNEW(I,J)
04310            TEMP1=C(ICONJ,J)+CNEW(ICONJ,J)
04320            IF(TEMP1.LE.TEMP)GO TO 52
04330            X(ICONJ,J)=0
04340            KLB=KLB-TEMP1+TEMP
04350            DO 55 K=1,P
04360            TE(I,K)=TE(I,K)+A(I,J,K)
04370    55      TE(ICONJ,K)=TE(ICONJ,K)-A(ICONJ,J,K)
04380            ICOST=ICOST-C(ICONJ,J)+C(I,J)
04390            CONE(J)=I
04400            X(I,J)=1
04410    52      CONTINUE
04420            GO TO 93
04430    130     FL(I,J)=.FALSE.
04440            TOP=TOP-1
04450            ICONJ=CONE(J)
```

```
0446          TEMP=CNEW(ICONJ,J)+C(ICONJ,J)
0447          TEMP1=C(I,J)+CNEW(I,J)
0448          IF(ITRACK)GO TO 42
0449          IF(TEMP1.GT.TEMP)GO TO 16
0450    42    X(ICONJ,J)=0
0451          X(I,J)=1
0452          KLB=KLB-TEMP+TEMP1
0453          ICOST=ICOST+C(I,J)-C(ICONJ,J)
0454          SE(J)=TEMP-TEMP1
0455          DO 124 K=1,P
0456          TE(ICONJ,K)=TE(ICONJ,K)-A(ICONJ,J,K)
0457    124   TE(I,K)=TE(I,K)+A(I,J,K)
0458          CONE(J)=I
0459          GO TO 16
0460    78    CALL RTIME(N5)
0461          ITOTAL=N5-N1
0462          WRITE(23,102)ITOTAL
0463    102   FORMAT(1X,'TOTAL TIME ',I10)
0464          WRITE(23,103)ICOU
0465    103   FORMAT(1X,' TOTAL NO OF NODES ',I7)
0466          RETURN;END
0467          SUBROUTINE GENRAT(M,N,P,A,C,B,ISEED)
0468    **    ************************************************************
0469    **    THIS SUBROUTINE GENERATES A MULTI-RESOURCE GENERALISED
0470    **    ASSIGNMENT PROBLEM
0471    **    ************************************************************
0472          DIMENSION A(25,200,6),C(25,200),B(25,6),SUM(25,5)
0473          INTEGER P
0474          CALL SEIRAN(ISEED)
0475          DO 1 I=1,M
0476          DO 1 K=1,P
0477    1     SUM(I,K)=0
0478          DO 10 J=1,N
0479          PMIN=100000.0
0480          DO 20 I=1,M
0481          DO 30 K=1,P
0482          PK=20*RAN(X)+5
0483    C     IPP=PK+0.99
0484          A(I,J,K)=PK
0485    30    CONTINUE
0486          PK=40*RAN(X)+10
0487          IPK=PK
0488          C(I,J)=IPK
0489          IF(C(I,J).GT.PMIN) GO TO 20
```

```
00490    20      CONTINUE
00500            DO 40 K=1,P
00510    40      SUM(INDEX,K)=SUM(INDEX,K)+A(INDEX,J,K)
00520    10      CONTINUE
00530            PMAX=0
00540            DO 50 I=1,M
00550            DO 50 K=1,P
00560            IF(SUM(I,K).LT.PMAX)GO TO 50
00570            PMAX=SUM(I,K)
00580    50      CONTINUE
00590            PB=0.6*N/M*15+0.4*PMAX
00600            DO 77 I=1,M
00610            DO 77 K=1,P
00620    77      B(I,K)=PB
00630            RETURN;END
```